**Министерство образования Российской Федерации**

**МОСКВОСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э.БАУМАНА**

Факультет: Информатика и системы управления (ИУ)  
Кафедра: Информационная безопасность (ИУ8)

**МЕТОДЫ ОПТИМИЗАЦИИ**

**Лабораторная работа №3 на тему:**   
«Целочисленное линейное программирование.

Метод ветвей и границ»

Вариант ­ 1

**Преподаватель:**   
Коннова Н.С.

**Студент:**   
Александров А.Н.

**Группа:**   
ИУ8-34

Москва, 2020

**Цель работы:**

изучить постановку задачи целочисленного линейного программирования (ЦЛП); овладеть навыками решения задач ЦЛП с помощью метода ветвей и границ (МВГ).

**Постановка задачи:**

Требуется найти решение следующей задачи:

Необходимо среди всех *3*-мерных векторов , удовлетворяющих системе

;

такой, для которого достигается минимум ЦФ:

.

Здесь:

**c** – вектор коэффициентов целевой функции(ЦФ),

**A** – матрица системы ограничений,  
**b** – вектор правой части системы ограничений.

**Ход работы:**

Исходная задача ЦЛП имеет вид:

Оптимальное решение задачи ЛП, полученное мной в лабораторной работе

Всего имеется 33 допустимых целочисленных решения:

(0, 0, 0);

(0, 0, 1);

(0, 1, 0);

(0, 1, 1);

(0, 2, 0);

(0, 2, 1);

(1, 0, 0);

(1, 0, 1);

(1, 1, 0);

(1, 1, 1);

(1, 2, 0);

(1, 2, 1);(2, 0, 0);

(2, 0, 1);

(2, 1, 0);

(2, 1, 1);

(2, 2, 0);

(2, 2, 1);(3, 0, 0);

(3, 0, 1);

(3, 1, 0);

(3, 1, 1);

(4, 0, 0);

(4, 0, 1);

(4, 1, 0);

(4, 1, 1);

(5, 0, 0);

(5, 0, 1);

(5, 1, 0);

(5, 1, 1);

(6, 0, 0);

(6, 0, 1);

(7, 0, 0)

Полный перебор всех вариантов даёт решение:

Для решения задачи ЦЛП методом ветвей и границ воспользовался дополненной версией собственной программы, написанной на языке программирования **Python**. Немного о дополнительных файлах проекта (см. Приложение А):

***1. brute\_force\_method.py***

В данном файле содержится класс ***BruteForceMethod***. Он реализует полный перебор всех целочисленных решений и нахождение лучшего среди них.

Поля класса ***BruteForceMethod***:

Поле **obj\_func\_coffs\_**: хранит коэффициенты целевой функции (ЦФ).

Поле **obj\_constraint\_system\_lhs\_**: хранит левую часть системы ограничений.

Поле **obj\_constraint\_system\_rhs\_**: хранит правую часть системы ограничений.

Поле **func\_direction\_**: хранит в себе направление целевой функции (поиск минимума или поиск максимума).

Поле **all\_solutions\_**: заполняется списком всевозможных значений аргументов, удовлетворяющих ограничениям.

Поля **max\_ind\_, max\_func\_value\_**: индекс максимального в списке **all\_solutions\_** и само значение функции соответственно.

***2. branch\_and\_bound.py***

В данном файле реализован класс ***BranchAndBound***, отвечающий за решение задачи ЦЛП методом ветвей и границ.

Поля класса ***BranchAndBound***:

Поле **obj\_func\_coffs\_**: хранит коэффициенты целевой функции (ЦФ).

Поле **obj\_constraint\_system\_lhs\_**: хранит левую часть системы ограничений.

Поле **obj\_constraint\_system\_rhs\_**: хранит правую часть системы ограничений.

Поле **func\_direction\_**: хранит в себе направление целевой функции (поиск минимума или поиск максимума).

Поле **simplex\_table\_**: хранит в себе начальную симплекс таблицу.

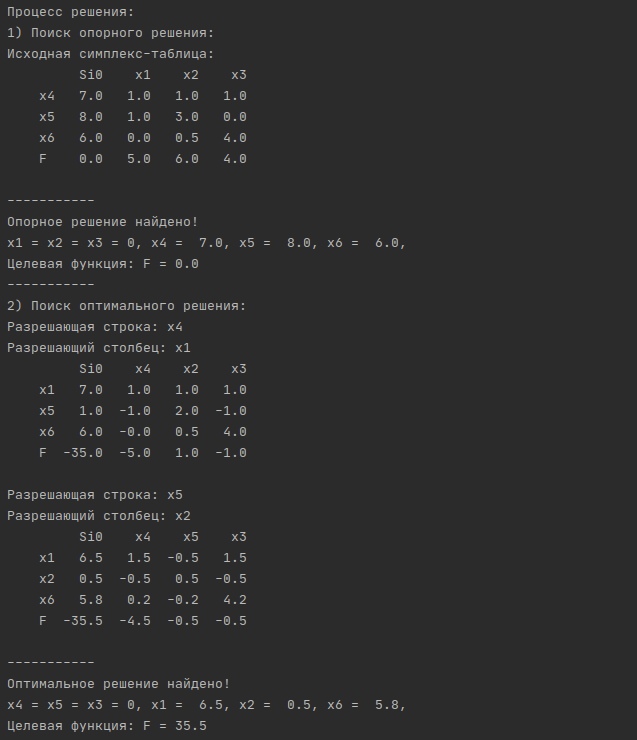
Поле **extra\_constraints\_lhs\_**: дополнительные ограничения, добавляемые в ходе ветвления по дробным переменным.

Поле **extra\_constraints\_rhs\_**: дополнительные ограничения, добавляемые в ходе ветвления по дробным переменным.

Поле **solutions\_storage\_**: хранилище целочисленных решений, полученных в ходе решения задачи методом ветвей и границ.

Решим исходную задачу ЦЛП:

Рисунок 1 Решение задачи ЛП симплекс-методом.



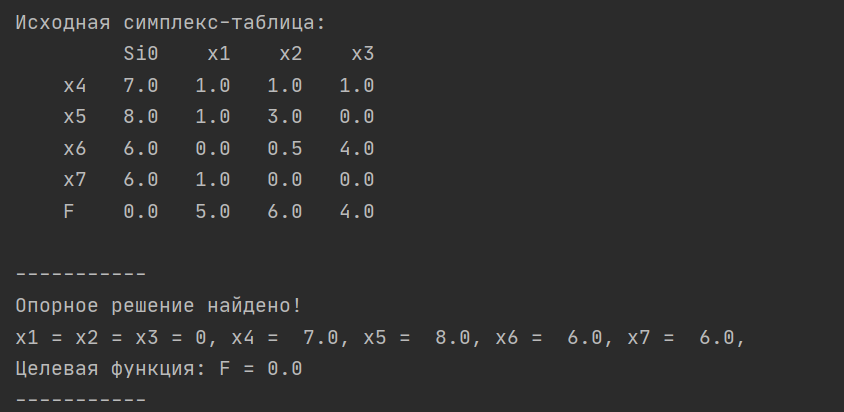
Решение задачи ЛП:

Осуществим ветвление по переменной .

1)Введём новое ограничение и решим задачу ЦЛП:

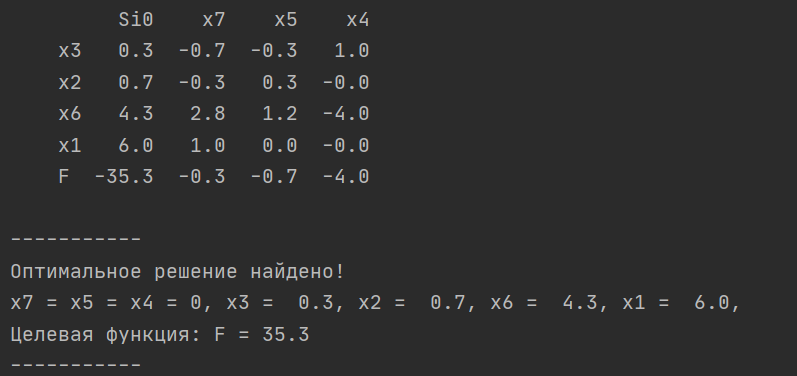
Соответствующая симплекс-таблица имеет вид:

Рисунок 2 Исходная симплекс-таблица.



После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 3 Конечная симплекс-таблица



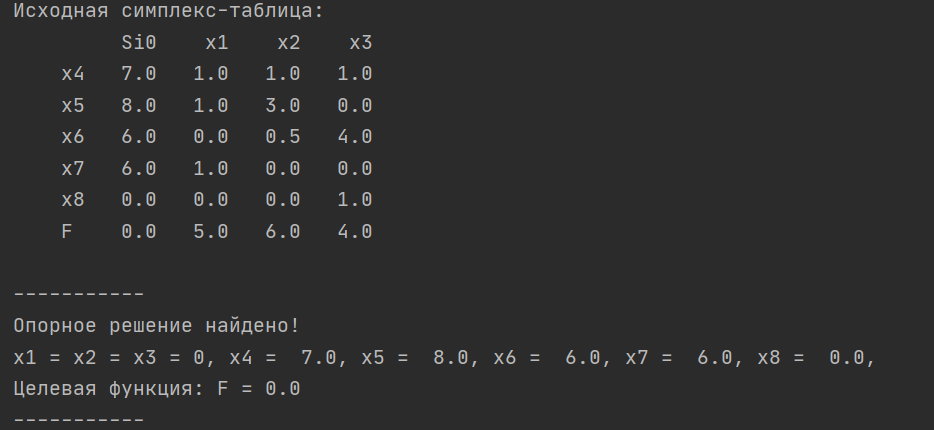
Полученное решение целочисленным не является.

Осуществим ветвление по переменной .

1.1)Введём новое ограничение и получим задачу ЦЛП:

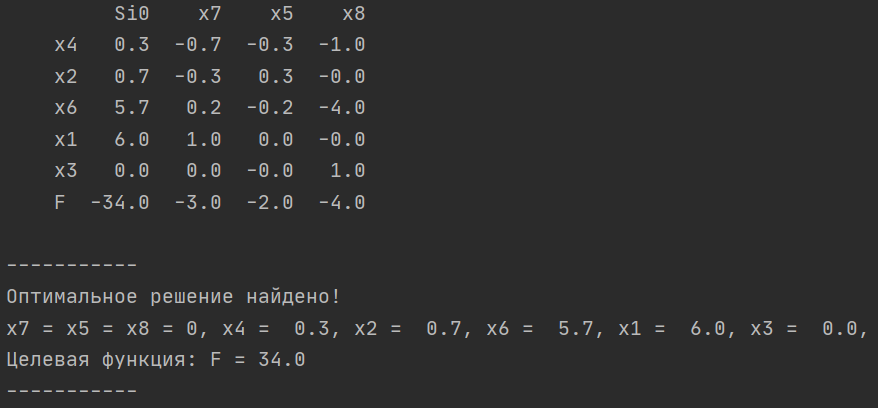
Соответствующая симплекс-таблица имеет вид:

Рисунок 4 Исходная симплекс-таблица.



После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 5 Конечная симплекс-таблица.



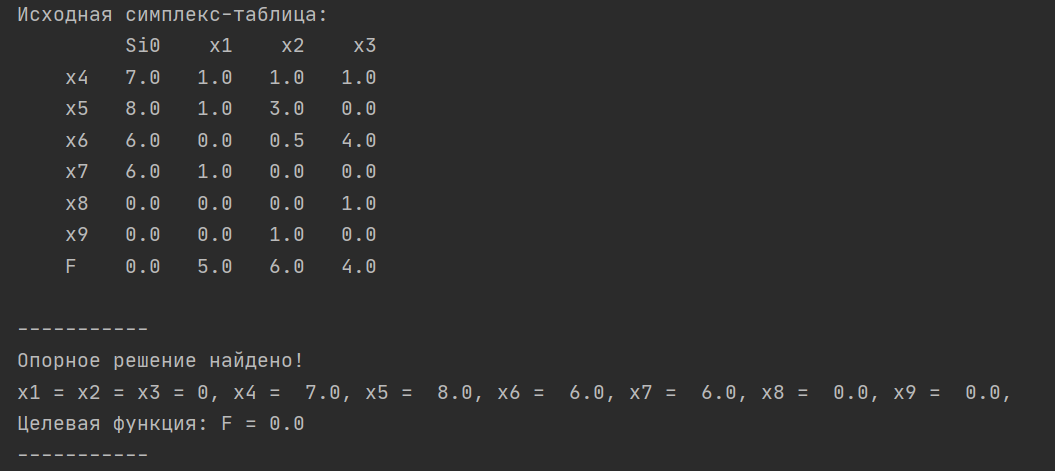
Полученное решение целочисленным не является.

Осуществим ветвление по переменной .

1.1.1)Введём новое ограничение и получим задачу ЦЛП:

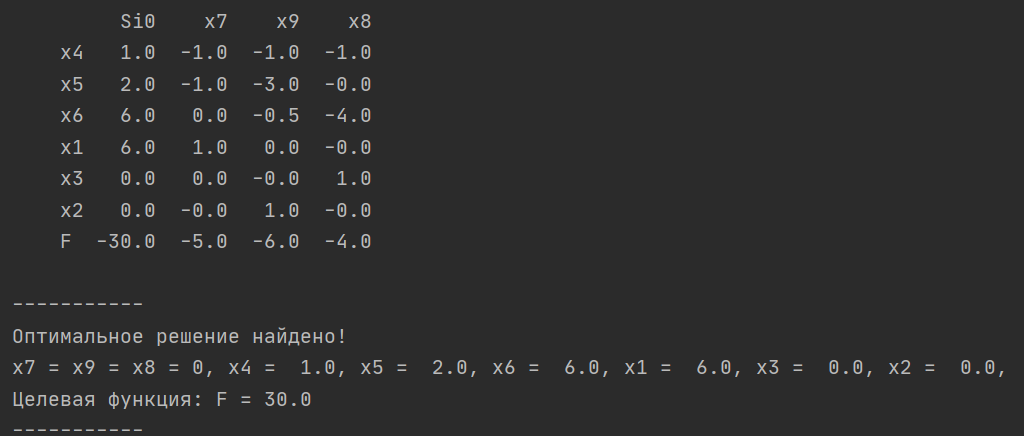
Соответствующая симплекс-таблица имеет вид:

Рисунок 6 Исходная симплекс-таблица.



После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 7 Конечная симплекс-таблица.



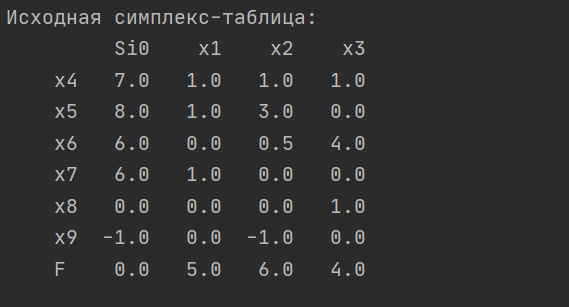
Получаем целочисленное решение:

.

1.1.2)Введём новое ограничение и получим задачу ЦЛП:

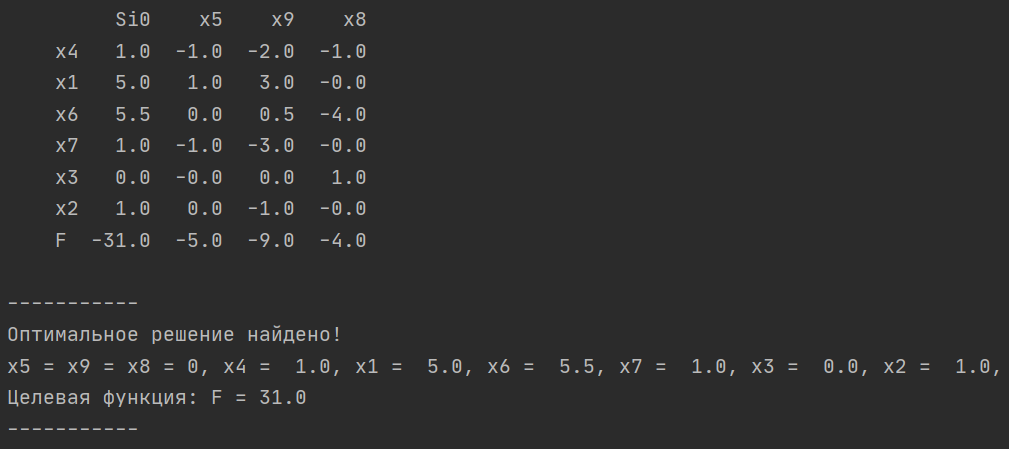
Соответствующая симплекс-таблица имеет вид:

Рисунок 8 Исходная симплекс-таблица.



После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 9 Конечная симплекс-таблица.



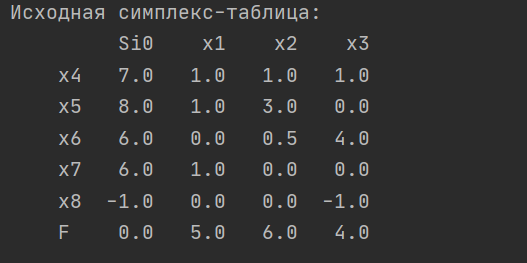
Получаем целочисленное решение:

.

1.2)Введём новое ограничение и получим задачу ЦЛП:

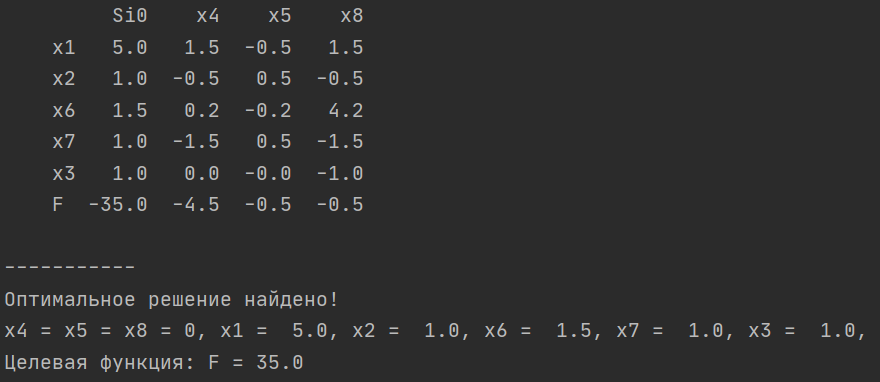
Соответствующая симплекс-таблица имеет вид:

Рисунок 10 Исходная симплекс-таблица.



После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок Конечная симплекс-таблица.



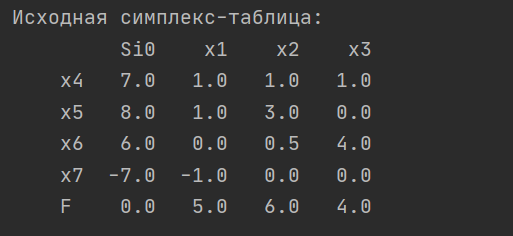
Получаем целочисленное решение:

.

1.2)Введём новое ограничение и получим задачу ЦЛП:

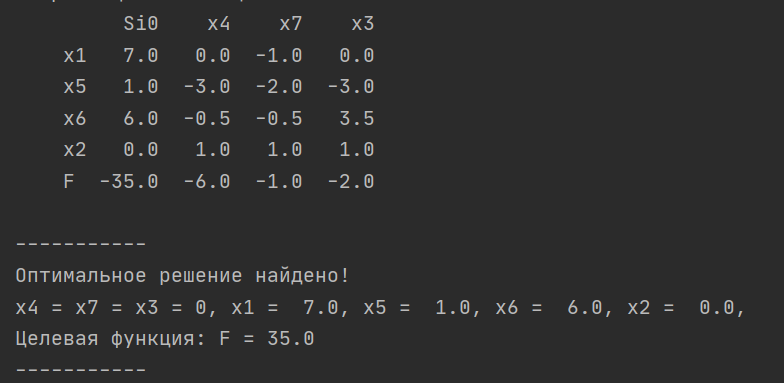
Соответствующая симплекс-таблица имеет вид:

Рисунок 12 Исходная симплекс-таблица.



После применения симплекс-метода конечная симплекс-таблица примет вид:

Рисунок 13 Конечная симплекс-таблица.

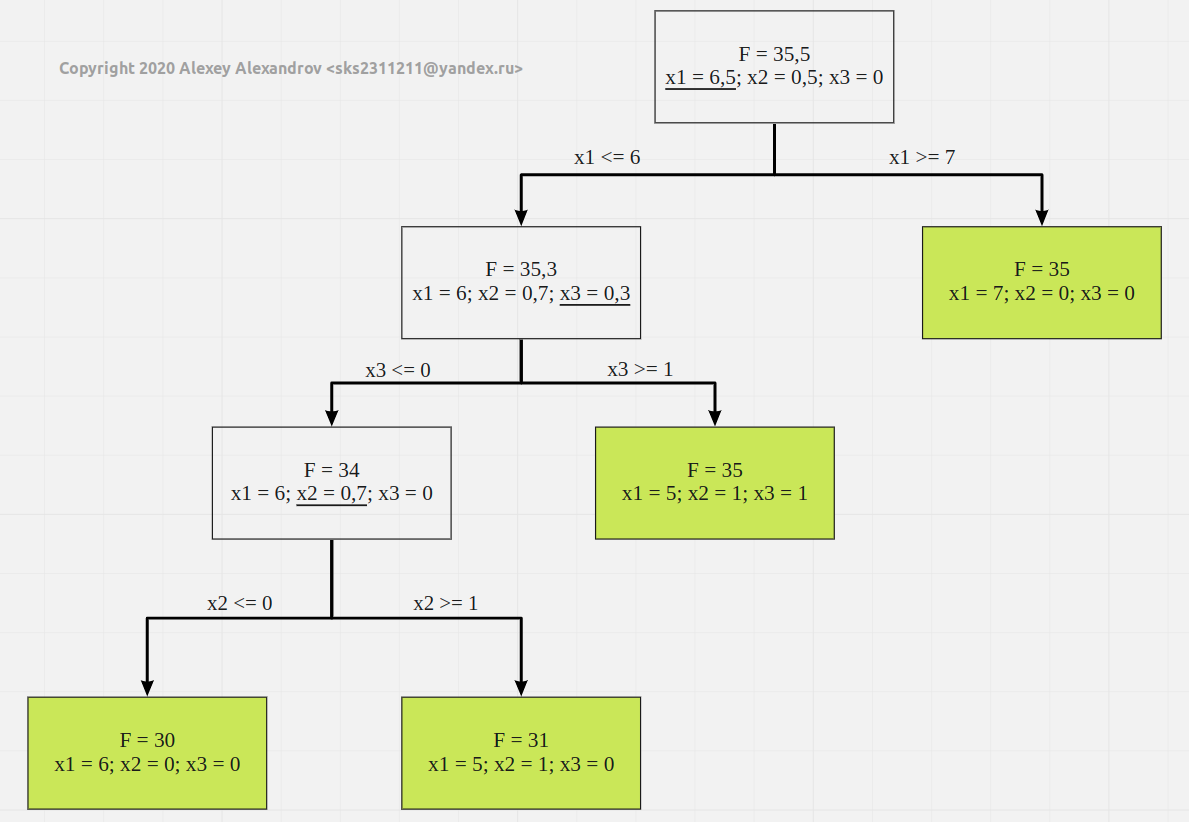


Получаем целочисленное решение:

.

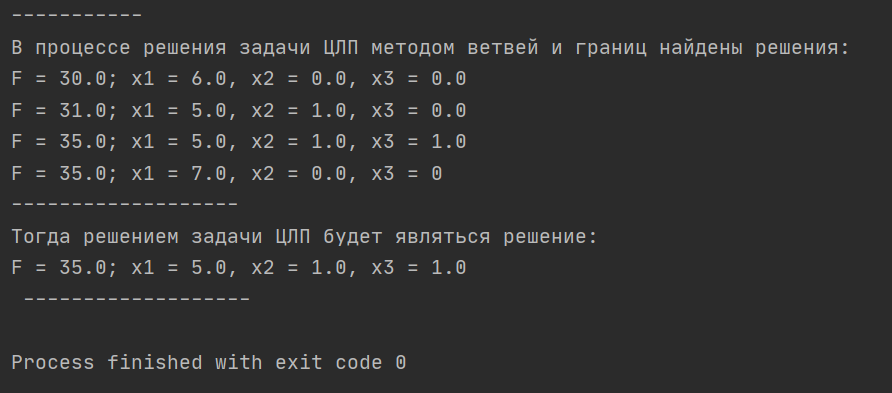
В итоге мы получили такое дерево решений:

Рисунок 14 Дерево решений задачи ЦЛП



Оптимальное решение:

Рисунок 15 Оптимальное решение задачи ЦЛП.



.

Проверка полученного решения:

**Вывод:**

В ходе проделанной работы я изучил постановку задачи целочисленного линейного программирования, получил навыки решения задачи ЦЛП методом ветвей и границ. Данный метод будет полезен для решения других задач.

МВГ является улучшенной версией метода полного перебора, так как на каждом шаге ветвления мы фактически отсекаем большинство ненужных нам значений переменных, и скорость решения задачи ЦЛП увеличивается.

**Приложение А**

*Код программы:*

***Файл “main.py”***

*# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>  
  
"""  
Лабораторная работа № 3  
Целочисленное линейное программирование. Метод ветвей и границ.  
Цель работы: изучить постановку задачи ЦЛП; получить навыки  
решения задачи ЦЛП методом ветвей и границ.  
  
Вариант 1.  
"""*from simplex import \*  
from branch\_and\_bound import \*  
import brute\_force\_method  
import branch\_and\_bound  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 problem = Simplex(**"input\_data.json"**)  
  
 print(problem)  
 *# Находим опорное решение задачи ЛП* problem.reference\_solution()  
 *# Находим оптимальное решение задачи ЛП* problem.optimal\_solution()  
  
 *# Находим рещение задачи ЦЛП полным перебором.* brute\_force = brute\_force\_method.BruteForceMethod(**"input\_data.json"**)  
 print(brute\_force)  
  
 *# Переходим к методу ветвей и границ.* bb = branch\_and\_bound.BranchAndBound(problem)  
 print(bb)

***Файл “simplex.py”***

*# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>*from simplex\_table import \*  
import json  
  
  
class Simplex:  
 *"""  
 Класс для решения задачи ЛП симплекс-методом.  
 """* def \_\_init\_\_(self, path\_to\_file):  
 *"""  
 Переопределённый метод \_\_init\_\_. Регистрирует входные данные из JSON-файла.  
 Определяем условие задачи.  
 :param path\_to\_file: путь до JSON-файла с входными данными.  
 """  
  
 # Парсим JSON-файл с входными данными* with open(path\_to\_file, **"r"**) as read\_file:  
 json\_data = json.load(read\_file)  
 self.obj\_func\_coffs\_ = np.array(json\_data[**"obj\_func\_coffs"**]) *# вектор-строка с - коэффициенты ЦФ* self.constraint\_system\_lhs\_ = np.array(json\_data[**"constraint\_system\_lhs"**]) *# матрица ограничений А* self.constraint\_system\_rhs\_ = np.array(json\_data[**"constraint\_system\_rhs"**]) *# вектор-столбец ограничений b* self.func\_direction\_ = json\_data[**"func\_direction"**] *# направление задачи (min или max)* if len(self.constraint\_system\_rhs\_) != self.constraint\_system\_rhs\_.shape[0]:  
 raise SimplexException(  
 **"Ошибка при вводе данных. Число строк в матрице и столбце ограничений не совпадает."**)  
  
 *# Если задача на max, то меняем знаки ЦФ и направление задачи (в конце возьмем решение со знаком минус и  
 # получим искомое).* if self.func\_direction\_ == **"max"**:  
 self.obj\_func\_coffs\_ \*= -1  
  
 *# Инициализация симплекс-таблицы.* self.simplex\_table\_ = SimplexTable(self.obj\_func\_coffs\_, self.constraint\_system\_lhs\_,  
 self.constraint\_system\_rhs\_)  
  
 def \_\_str\_\_(self):  
 *"""  
 Переопренный метод \_\_str\_\_ для условия задачи.  
 :return: Строка с выводом условия задачи.  
 """* output = **"""Условие задачи:  
------------------------------------------------------------  
Найти вектор x = (x1,x2,..., xn)^T как решение след. задачи:"""** output += **f"**\n**F = cx ->** {self.func\_direction\_}**,"** output += **"**\n**Ax <= b,**\n**x1,x2, ..., xn >= 0"** output += **f"**\n**C =** {self.obj\_func\_coffs\_}**,"** output += **f"**\n**A =**\n{self.constraint\_system\_lhs\_}**,"** output += **f"**\n**b^T =** {self.constraint\_system\_rhs\_}**."** output += **"**\n**------------------------------------------------------------"** return output  
  
 *# Этап 1. Поиск опорного решения.* def reference\_solution(self):  
 *"""  
 Метод производит отыскание опорного решения.  
 """* print(**"Процесс решения:**\n**1) Поиск опорного решения:"**)  
 print(**"Исходная симплекс-таблица:"**, self.simplex\_table\_, sep=**"**\n**"**)  
  
 while not self.simplex\_table\_.is\_find\_ref\_solution():  
 self.simplex\_table\_.search\_ref\_solution()  
  
 print(**"-----------"**)  
 print(**"Опорное решение найдено!"**)  
 self.output\_solution()  
 print(**"-----------"**)  
  
 *# Этап 2. Поиск оптимального решения.* def optimal\_solution(self):  
 *"""  
 Метод производит отыскание оптимального решения.  
 """* print(**"2) Поиск оптимального решения:"**)  
  
 while not self.simplex\_table\_.is\_find\_opt\_solution():  
 self.simplex\_table\_.optimize\_ref\_solution()  
  
 *# Если задача на max, то в начале свели задачу к поиску min, а теперь  
 # возьмём это решение со знаком минус и получим ответ для мак.* if self.func\_direction\_ == **"max"**:  
 self.simplex\_table\_.main\_table\_[self.simplex\_table\_.main\_table\_.shape[0] - 1][0] \*= -1  
  
 print(**"-----------"**)  
 print(**"Оптимальное решение найдено!"**)  
 self.output\_solution()  
 print(**"-----------"**)  
  
 def output\_solution(self):  
 *"""  
 Метод выводит текущее решение, используется для вывода опорного и потимального решений.  
 """* fict\_vars = self.simplex\_table\_.top\_row\_[2:]  
 last\_row\_ind = self.simplex\_table\_.main\_table\_.shape[0] - 1  
  
 for var in fict\_vars:  
 print(var, **"= "**, end=**""**)  
 print(0, end=**", "**)  
  
 for i in range(last\_row\_ind):  
 print(self.simplex\_table\_.left\_column\_[i], **"= "**, round(self.simplex\_table\_.main\_table\_[i][0], 1), end=**", "**)  
  
 print(**"**\n**Целевая функция: F ="**, round(self.simplex\_table\_.main\_table\_[last\_row\_ind][0], 1))

***Файл “simplex\_table.py”***

*# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>*import numpy as np  
  
  
class SimplexException(Exception):  
 *"""Пользовательское исключение для решения задач симплекс-методом."""* def \_\_init\_\_(self, text):  
 self.txt = text  
  
  
class SimplexTable:  
 *"""  
 Класс симплекс-таблицы.  
 """* def \_\_init\_\_(self, obj\_func\_coffs, constraint\_system\_lhs, constraint\_system\_rhs):  
 *"""  
 Переопределённый метод \_\_init\_\_ для создания экземпляра класса SimplexTable.  
 :param obj\_func\_coffs: коэффициенты ЦФ.  
 :param constraint\_system\_lhs: левая часть системы ограничений.  
 :param constraint\_system\_rhs: правая часть системы ограничений.  
 """* var\_count = len(obj\_func\_coffs)  
 constraint\_count = constraint\_system\_lhs.shape[0]  
  
 *# Заполнение верхнего хедера.* self.top\_row\_ = [**" "**, **"Si0"**]  
 for i in range(var\_count):  
 self.top\_row\_.append(**"x"** + str(i + 1))  
  
 *# Заполнение левого хедера.* self.left\_column\_ = []  
 ind = var\_count  
 for i in range(constraint\_count):  
 ind += 1  
 self.left\_column\_.append(**"x"** + str(ind))  
 self.left\_column\_.append(**"F "**)  
  
 self.main\_table\_ = np.zeros((constraint\_count + 1, var\_count + 1))  
 *# Заполняем столбец Si0.* for i in range(constraint\_count):  
 self.main\_table\_[i][0] = constraint\_system\_rhs[i]  
 *# Заполняем строку F.* for j in range(var\_count):  
 self.main\_table\_[constraint\_count][j + 1] = -obj\_func\_coffs[j]  
  
 *# Заполняем А.* for i in range(constraint\_count):  
 for j in range(var\_count):  
 self.main\_table\_[i][j + 1] = constraint\_system\_lhs[i][j]  
  
 def \_\_str\_\_(self):  
 *"""  
 Переопренный метод \_\_str\_\_ для симплекс-таблицы.  
 :return: Строка с выводом симплекс-таблицы.  
 """* output = **""** for header in self.top\_row\_:  
 output += **'{:>6}'**.format(header)  
 output += **"**\n**"** for i in range(self.main\_table\_.shape[0]):  
 output += **'{:>6}'**.format(self.left\_column\_[i])  
 for j in range(self.main\_table\_.shape[1]):  
 output += **'{:>6}'**.format(round(self.main\_table\_[i][j], 1))  
 output += **"**\n**"** return output  
  
 def is\_find\_ref\_solution(self):  
 *"""  
 Функция проверяет, найдено ли опорное решение по свободным в симплекс-таблице.  
 :return: True - опорное решение уже найдено. False - полученное решение пока не является опорным.  
 """  
  
 # Проверяем все, кроме коэффициента ЦФ* for i in range(self.main\_table\_.shape[0] - 1):  
 if self.main\_table\_[i][0] < 0:  
 return False  
 return True  
  
 def search\_ref\_solution(self):  
 *"""  
 Функция производит одну итерацию поиска опорного решения.  
 """* res\_row = None  
 for i in range(self.main\_table\_.shape[0] - 1):  
 if self.main\_table\_[i][0] < 0:  
 res\_row = i  
 break  
  
 *# Если найден отрицательный элемент в столбце свободных членов, то ищем первый отрицательный в строке с ней.* res\_col = None  
 if res\_row is not None:  
 for j in range(1, self.main\_table\_.shape[1]):  
 if self.main\_table\_[res\_row, j] < 0:  
 res\_col = j  
 break  
  
 *# Если найден разрешающий столбец, то находим в нём разрешающий элемент.* res\_element = None  
 if res\_col is not None:  
 *# Ищем минимальное положительное отношение Si0 / x[res\_col]* minimum = None  
 ind = -1  
 *# for i in range(self.main\_table\_.shape[0] - 1):  
 # curr = self.main\_table\_[i][0] / self.main\_table\_[i][res\_col]  
 # if curr <= 0 or (self.main\_table\_[i][res\_col] == 0):  
 # continue  
 # elif minimum is None:  
 # minimum = curr  
 # ind = i  
 # elif curr < minimum:  
 # minimum = curr  
 # ind = i* for i in range(self.main\_table\_.shape[0] - 1):  
 *# Ищем минимальное отношение -- разрешающую строку.* curr = self.main\_table\_[i][res\_col]  
 s\_i0 = self.main\_table\_[i][0]  
 if curr == 0:  
 continue  
 elif (s\_i0 / curr) > 0 and (minimum is None or (s\_i0 / curr) < minimum):  
 minimum = (s\_i0 / curr)  
 ind = i  
  
 if minimum is None:  
 raise SimplexException(**"Решения не существует! При нахождении опорного решения не нашлось минимального "  
 "положительного отношения."**)  
 else:  
 res\_row = ind  
  
 *# Разрешающий элемент найден.* res\_element = self.main\_table\_[res\_row][res\_col]  
 print(**"Разрешающая строка: {}"**.format(self.left\_column\_[res\_row]))  
 print(**"Разрешающий столбец: {}"**.format(self.top\_row\_[res\_col + 1]))  
  
 *# Пересчёт симплекс-таблицы.* self.recalc\_table(res\_row, res\_col, res\_element)  
 else:  
 raise SimplexException(**"Задача не имеет допустимых решений! При нахождении опорного решения не нашлось "  
 "отрицательного элемента в строке с отрицательным свободным членом."**)  
  
 def is\_find\_opt\_solution(self):  
 *"""  
 Функция проверяет, найдено ли оптимальное решение по коэффициентам ЦФ в симплекс-таблице.  
 :return: True - оптимальное решение уже найдено. False - полученное решение пока не оптимально.  
 """* for i in range(1, self.main\_table\_.shape[1]):  
 if self.main\_table\_[self.main\_table\_.shape[0] - 1][i] > 0:  
 return False  
 *# Если положительных не нашлось, то оптимальное решение уже найдено.* return True  
  
 def optimize\_ref\_solution(self):  
 *"""  
 Функция производит одну итерацию поиска оптимального решения на основе  
 уже полученного опорного решения.  
 """* res\_col = None  
 ind\_f = self.main\_table\_.shape[0] - 1  
  
 *# В строке F ищем первый положительный.* for j in range(1, self.main\_table\_.shape[1]):  
 curr = self.main\_table\_[ind\_f][j]  
 if curr > 0:  
 res\_col = j  
 break  
  
 minimum = None  
 res\_row = None  
  
 *# Идём по всем, кроме ЦФ ищём минимальное отношение.* for i in range(self.main\_table\_.shape[0] - 1):  
 *# Ищем минимальное отношение -- разрешающую строку.* curr = self.main\_table\_[i][res\_col]  
 s\_i0 = self.main\_table\_[i][0]  
 if curr < 0:  
 continue  
 elif (s\_i0 / curr) >= 0 and (minimum is None or (s\_i0 / curr) < minimum):  
 minimum = (s\_i0 / curr)  
 res\_row = i  
  
 if res\_row is None:  
 raise SimplexException(**"Функция не ограничена! Оптимального решения не существует."**)  
  
 else:  
 *# Разрешающий элемент найден.* res\_element = self.main\_table\_[res\_row][res\_col]  
 print(**"Разрешающая строка: {}"**.format(self.left\_column\_[res\_row]))  
 print(**"Разрешающий столбец: {}"**.format(self.top\_row\_[res\_col + 1]))  
 *# Пересчёт симплекс-таблицы.* self.recalc\_table(res\_row, res\_col, res\_element)  
  
 def recalc\_table(self, res\_row, res\_col, res\_element):  
 *"""  
 Функция по заданным разрешающим строке, столбцу и элекменту производит перерасчёт  
 симплекс-таблицы методом жордановых искоючений.  
 :param res\_row: индекс разрешающей строки  
 :param res\_col: индекс разрешающего столбца  
 :param res\_element: разрешающий элемент  
 """* recalced\_table = np.zeros((self.main\_table\_.shape[0], self.main\_table\_.shape[1]))  
  
 *# Пересчёт разрешающего элемента.* recalced\_table[res\_row][res\_col] = 1 / res\_element  
  
 *# Пересчёт разрешающей строки.* for j in range(self.main\_table\_.shape[1]):  
 if j != res\_col:  
 recalced\_table[res\_row][j] = self.main\_table\_[res\_row][j] / res\_element  
  
 *# Пересчёт разрешающего столбца.* for i in range(self.main\_table\_.shape[0]):  
 if i != res\_row:  
 recalced\_table[i][res\_col] = -(self.main\_table\_[i][res\_col] / res\_element)  
  
 *# Пересчёт оставшейся части таблицы.* for i in range(self.main\_table\_.shape[0]):  
 for j in range(self.main\_table\_.shape[1]):  
 if (i != res\_row) and (j != res\_col):  
 recalced\_table[i][j] = self.main\_table\_[i][j] - (  
 (self.main\_table\_[i][res\_col] \* self.main\_table\_[res\_row][j]) / res\_element)  
  
 self.main\_table\_ = recalced\_table  
  
 self.swap\_headers(res\_row, res\_col)  
  
 print(self.\_\_str\_\_())  
  
 def swap\_headers(self, res\_row, res\_col):  
 *"""  
 Функция меняет меняет переменные в строке и столбце местами.  
 :param res\_row: разрешающая строка  
 :param res\_col: разрешающий столбец  
 """* temp = self.top\_row\_[res\_col + 1]  
 self.top\_row\_[res\_col + 1] = self.left\_column\_[res\_row]  
 self.left\_column\_[res\_row] = temp

***Файл “branch\_and\_bound.py”***

*# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>*import math  
  
from simplex\_b\_b import \*  
  
  
class BranchAndBound:  
 *"""Класс задачи ветвей и границ."""* def \_\_init\_\_(self, simplex\_problem):  
 *"""  
 Переопределённый метод \_\_init\_\_. Определяем условие задачи на основе задачи ЛП.  
 :param simplex\_problem: задача ЛП.  
 """* self.obj\_func\_coffs\_ = simplex\_problem.obj\_func\_coffs\_ *# вектор-строка с - коэффициенты ЦФ.* self.constraint\_system\_lhs\_ = simplex\_problem.constraint\_system\_lhs\_ *# матрица ограничений А.* self.constraint\_system\_rhs\_ = simplex\_problem.constraint\_system\_rhs\_ *# вектор-столбец ограничений b.* self.func\_direction\_ = simplex\_problem.func\_direction\_ *# направление задачи (min или max).* self.simplex\_table\_ = simplex\_problem.simplex\_table\_ *# симплекс-таблица* self.extra\_constraints\_lhs\_ = [] *# дополнительные строки ограничений для матрицы А.* self.extra\_constraints\_rhs\_ = [] *# дополнительные элементы ограничений для столбца b.* self.solutions\_storage\_ = [] *# хранилише целочисленных решений, найденных в ходе ветвления.* if not is\_integer\_solution(self.simplex\_table\_):  
 self.branching(self.simplex\_table\_, self.extra\_constraints\_lhs\_, self.extra\_constraints\_rhs\_)  
  
 def \_\_str\_\_(self):  
 *"""  
 Переопренный метод \_\_str\_\_ для вывода задачи ЦЛП методом ветвей и границ.  
 :return: Строка с выводом задачи.  
 """* output = **"В процессе решения задачи ЦЛП методом ветвей и границ найдены решения:**\n**"** for solution in self.solutions\_storage\_:  
 output += **f"F =** {solution[0]}**; x1 =** {solution[1][0]}**, x2 =** {solution[1][1]}**, x3 =** {solution[1][2]}\n**"** best\_solution = self.find\_best\_solution()  
 output += **"-------------------**\n**"** output += **f"Тогда решением задачи ЦЛП будет являться решение:**\n**"** \  
 **f"F =** {best\_solution[0]}**; "** \  
 **f"x1 =** {best\_solution[1][0]}**, "** \  
 **f"x2 =** {best\_solution[1][1]}**, "** \  
 **f"x3 =** {best\_solution[1][2]}\n **"** output += **"-------------------"** return output  
  
 def branching(self, simplex\_table, extra\_constraints\_lhs, extra\_constraints\_rhs):  
 *"""  
 Метод производит ветвление на основе текущей симплекс таблицы.  
 :param simplex\_table: текущая симплекс таблица.  
 :param extra\_constraints\_lhs: дополнительные ограничения в левой части системы.  
 :param extra\_constraints\_rhs: дополнительные ограничения в правой части системы.  
 """  
  
 # В столбце свободных членов симплекс таблицы ищем нецелую и нефиктивную переменную.* b\_var\_number, b\_var\_value = branch\_var\_search(simplex\_table)  
 constraint\_system\_lhs = self.constraint\_system\_lhs\_  
 constraint\_system\_rhs = self.constraint\_system\_rhs\_  
  
 print(**f"Осуществим ветвление по переменной x**{b\_var\_number} **=** {round(b\_var\_value, 1)}**"**)  
 print(  
 **f"x**{b\_var\_number} **<=** {math.floor(b\_var\_value)} **и x**{b\_var\_number} **>=** {math.ceil(b\_var\_value)} **"  
 f"(-x**{b\_var\_number} **<=** {-math.ceil(b\_var\_value)}**)"**)  
  
 *# Строка ограничений для ветвления x <= [x\*] (левая ветка)  
  
 # Заготавливаем строку ограничений для левой части системы.* lhs\_constraint\_temp\_row = np.zeros(3)  
 lhs\_constraint\_temp\_row[b\_var\_number - 1] = 1  
  
 extra\_constraints\_lhs.append(lhs\_constraint\_temp\_row)  
 extra\_constraints\_rhs.append(math.floor(b\_var\_value))  
  
 for i in range(len(extra\_constraints\_rhs)):  
 *# Добавляем строку в левую часть системы ограничений.* constraint\_system\_lhs = np.vstack((constraint\_system\_lhs, extra\_constraints\_lhs[i]))  
 *# Добавляем элемент в правую часть системы ограничений.* constraint\_system\_rhs = np.append(constraint\_system\_rhs, extra\_constraints\_rhs[i])  
  
 *# Оборачиваем в try-except: если поднимется исключение в задаче, решение пойдёт дальше по другим веткам и узлам.* try:  
 left\_branch\_task = SimplexBB(self.obj\_func\_coffs\_, constraint\_system\_lhs,  
 constraint\_system\_rhs,  
 self.func\_direction\_)  
  
 print(**f"Ограничение: x**{b\_var\_number} **<=** {math.floor(b\_var\_value)}**"**)  
 left\_branch\_task.reference\_solution()  
 left\_branch\_task.optimal\_solution()  
  
 if not is\_integer\_solution(left\_branch\_task.simplex\_table\_):  
 print(**"От левой ветки ветвимся дальше..."**)  
 self.branching(left\_branch\_task.simplex\_table\_, extra\_constraints\_lhs, extra\_constraints\_rhs)  
 else:  
 self.add\_solution(left\_branch\_task.simplex\_table\_)  
  
 except SimplexException as err:  
 print(  
 **f"При ветвлении по x**{b\_var\_number} **<=** {math.floor(b\_var\_value)} **Решение не было найдено:**\n{err}**"**)  
  
 constraint\_system\_lhs = np.delete(constraint\_system\_lhs, constraint\_system\_lhs.shape[0] - 1, axis=0)  
 constraint\_system\_rhs = np.delete(constraint\_system\_rhs, constraint\_system\_rhs.shape[0] - 1, axis=0)  
 extra\_constraints\_lhs = extra\_constraints\_lhs[:-1]  
 extra\_constraints\_rhs = extra\_constraints\_rhs[:-1]  
  
 *# Строка ограничений для ветвления x >= [x\*] + 1 (правая ветка)  
  
 # Заготавливаем строку ограничений для левой части системы.* lhs\_constraint\_temp\_row = np.zeros(3)  
 lhs\_constraint\_temp\_row[b\_var\_number - 1] = -1  
  
 extra\_constraints\_lhs.append(lhs\_constraint\_temp\_row)  
 extra\_constraints\_rhs.append(-math.ceil(b\_var\_value))  
 *# Добавляем строку в левую часть системы ограничений.* constraint\_system\_lhs = np.vstack((constraint\_system\_lhs, lhs\_constraint\_temp\_row))  
 *# Добавляем элемент в правую часть системы ограничений.* constraint\_system\_rhs = np.append(constraint\_system\_rhs, -math.ceil(b\_var\_value))  
  
 *# Оборачиваем в try-except: если поднимется исключение в задаче, решение пойдёт дальше по другим веткам и узлам.* try:  
 right\_branch\_task = SimplexBB(self.obj\_func\_coffs\_, constraint\_system\_lhs,  
 constraint\_system\_rhs,  
 self.func\_direction\_)  
 print(**f"Ограничение: x**{b\_var\_number} **>=** {math.ceil(b\_var\_value)}**"**)  
 right\_branch\_task.reference\_solution()  
 right\_branch\_task.optimal\_solution()  
  
 if not is\_integer\_solution(right\_branch\_task.simplex\_table\_):  
 print(**"От правой ветки ветвимся дальше..."**)  
 self.branching(right\_branch\_task.simplex\_table\_, extra\_constraints\_lhs, extra\_constraints\_rhs)  
  
 if is\_integer\_solution(right\_branch\_task.simplex\_table\_):  
 self.add\_solution(right\_branch\_task.simplex\_table\_)  
 return  
  
 except SimplexException as err:  
 print(  
 **f"При ветвлении по x**{b\_var\_number} **>=** {math.ceil(b\_var\_value)} **Решение не было найдено:**\n{err}**"**)  
  
 def find\_best\_solution(self):  
 *"""Метод производит отыскание наилучшего целочисленного решения среди полученных в ходе ветвлений."""* if self.func\_direction\_ == **"max"**:  
 maximum\_solution = None  
 for solution in self.solutions\_storage\_:  
 if maximum\_solution is None or maximum\_solution[0] < solution[0]:  
 maximum\_solution = solution  
  
 return maximum\_solution  
 else:  
 minimum\_solution = None  
 for solution in self.solutions\_storage\_:  
 if minimum\_solution is None or minimum\_solution[0] > solution[0]:  
 minimum\_solution = solution  
  
 return minimum\_solution  
  
 def add\_solution(self, simplex\_table):  
 *"""Метод добавляет целочисленное решение в self.solutions\_storage\_."""* rows\_num = simplex\_table.main\_table\_.shape[0]  
  
 f = simplex\_table.main\_table\_[rows\_num - 1][0]  
  
 vars\_values = [0, 0, 0]  
  
 for i in range(rows\_num - 1):  
 if simplex\_table.left\_column\_[i] in [**"x1"**, **"x2"**, **"x3"**]:  
 vars\_values[int(simplex\_table.left\_column\_[i][1]) - 1] = simplex\_table.main\_table\_[i][0]  
  
 self.solutions\_storage\_.append((f, vars\_values))  
  
  
def is\_integer\_solution(simplex\_table):  
 *"""  
 Проверяет, является ли найденное решение целовчисленным.  
 :param simplex\_table: текущее состояние задачи ЦЛП.  
 return: True - решение задачи ЛП целочисленно. False - не целочисленно.  
 """* for i in range(simplex\_table.main\_table\_.shape[0]):  
 if (simplex\_table.left\_column\_[i] in [**"x1"**, **"x2"**, **"x3"**]) and (not simplex\_table.main\_table\_[i][0].is\_integer()):  
 return False  
  
 return True  
  
  
def branch\_var\_search(simplex\_table):  
 *"""  
 Производит поиск переменной ветвления в симплекс-таблице.  
 :param simplex\_table: текущее состояние задачи ЦЛП.  
 return: возвращает номер переменной ветвления и её значение в симплекс-таблице.  
 """* for i in range(len(simplex\_table.left\_column\_) - 1):  
 if (not simplex\_table.main\_table\_[i][0].is\_integer()) and (simplex\_table.left\_column\_[i] in [**"x1"**, **"x2"**, **"x3"**]):  
 return int(simplex\_table.left\_column\_[i][1]), simplex\_table.main\_table\_[i][0]

***Файл “brute\_force\_method.json”***

*# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>*import json  
import numpy as np  
  
  
class BruteForceMethod:  
 *"""Класс для решения задачи ЦЛП путём полного перебора."""* def \_\_init\_\_(self, path\_to\_file):  
 *"""  
 Переопределённый метод \_\_init\_\_. Регистрирует входные данные из JSON-файла.  
 Определяем условие задачи.  
 :param path\_to\_file: путь до JSON-файла с входными данными.  
 """  
  
 # Парсим JSON-файл с входными данными* with open(path\_to\_file, **"r"**) as read\_file:  
 json\_data = json.load(read\_file)  
 self.obj\_func\_coffs\_ = np.array(json\_data[**"obj\_func\_coffs"**]) *# вектор-строка с - коэффициенты ЦФ* self.constraint\_system\_lhs\_ = np.array(json\_data[**"constraint\_system\_lhs"**]) *# матрица ограничений А* self.constraint\_system\_rhs\_ = np.array(json\_data[**"constraint\_system\_rhs"**]) *# вектор-столбец ограничений b* self.func\_direction\_ = json\_data[**"func\_direction"**] *# направление задачи (min или max)* if len(self.constraint\_system\_rhs\_) != self.constraint\_system\_rhs\_.shape[0]:  
 raise Exception(**"Ошибка при вводе данных. Число строк в матрице и столбце ограничений не совпадает."**)  
  
 if len(self.constraint\_system\_rhs\_) > len(self.obj\_func\_coffs\_):  
 raise Exception(**"СЛАУ несовместна! Число уравнений больше числа переменных."**)  
  
 self.all\_solutions\_ = self.list\_of\_solutions()  
 self.max\_ind\_, self.max\_func\_value\_ = self.search\_optimal\_solution()  
  
 def \_\_str\_\_(self):  
 *"""  
 Переопренный метод \_\_str\_\_ для вывода всех целочисленных решений, включая наилучшее.  
 :return: Строка с выводом всех решений и наилучшего из них.  
 """* output = **"Метод полного перебора.**\n**"** output += **f"Всего имеется** {len(self.all\_solutions\_)} **допустимых целочисленных решений:**\n**"** for solution in self.all\_solutions\_:  
 output += **f"(**{solution[0]}**,** {solution[1]}**,** {solution[2]}**);**\n**"** output += **"-----------**\n**"** output += **"Полный перебор всех решений даёт решение:**\n**"** output += **f"x1 =** {self.all\_solutions\_[self.max\_ind\_][0]}**,"** \  
 **f" x2 =** {self.all\_solutions\_[self.max\_ind\_][1]}**,"** \  
 **f" x3 =** {self.all\_solutions\_[self.max\_ind\_][2]}**,"** \  
 **f" F =** {self.max\_func\_value\_} \n**"** output += **"-----------**\n**"** return output  
  
 def is\_satisfies\_constraints(self, x1, x2, x3):  
 *"""  
 Метод проверяет, удовлетворяет ли заданный набор значений переменных ограничениям.  
 :return: True -- удовлетворяет ограничениям, False -- не удовлетворяет им.  
 """* for i in range(len(self.constraint\_system\_rhs\_)):  
 if self.constraint\_system\_lhs\_[i][0] \* x1 + \  
 self.constraint\_system\_lhs\_[i][1] \* x2 + \  
 self.constraint\_system\_lhs\_[i][2] \* x3 > self.constraint\_system\_rhs\_[i]:  
 return False  
  
 return True  
  
 def list\_of\_solutions(self):  
 *"""  
 Перебирает все возможные целочисленные значения переменных и добавляет их в список.  
 :return: Список решений задачи ЦЛП.  
 """* maximum = 0  
 list\_of\_solutions = []  
  
 *# Находим максимальное в столбце ограничений* for i in range(len(self.constraint\_system\_lhs\_)):  
 if self.constraint\_system\_rhs\_[i] > maximum:  
 maximum = self.constraint\_system\_rhs\_[i]  
  
 for x1 in range(maximum + 1):  
 for x2 in range(maximum + 1):  
 for x3 in range(maximum + 1):  
 if self.is\_satisfies\_constraints(x1, x2, x3):  
 list\_of\_solutions.append(np.array([x1, x2, x3]))  
  
 return list\_of\_solutions  
  
 def search\_optimal\_solution(self):  
 *"""  
 В списке всех целочисленных решений производит отыскание максимального.  
 :return: возвращает индекс решения задачи ЦЛП в списке целочисленных решений и сам набор значений переменных.  
 """* list\_of\_func\_values = []  
  
 *# Получаем список* for i in range(len(self.all\_solutions\_)):  
 f = 0  
 for j in range(len(self.obj\_func\_coffs\_)):  
 f += self.obj\_func\_coffs\_[j] \* self.all\_solutions\_[i][j]  
 list\_of\_func\_values.append(f)  
  
 *# В этом списке ищем максимальное значение ЦФ и находим её индекс.* max\_f\_ind = 0  
 for i in range(len(list\_of\_func\_values)):  
 if list\_of\_func\_values[i] > list\_of\_func\_values[max\_f\_ind]:  
 max\_f\_ind = i  
  
 return max\_f\_ind, list\_of\_func\_values[max\_f\_ind]

***Файл “input\_data.json”***

{

"obj\_func\_coffs": [5, 6, 4],

"constraint\_system\_lhs": [

[1, 1, 1],

[1, 3, 0],

[0, 0.5, 4]

],

"constraint\_system\_rhs": [7, 8, 6],

"func\_direction": "max"

}